QUANTUM CONVOLUTIONAL STABILIZER CODES

A Thesis

by

NEELIMA CHINTHAMANI

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2004

Major Subject: Computer Science

QUANTUM CONVOLUTIONAL STABILIZER CODES

A Thesis

by

NEELIMA CHINTHAMANI

Submitted to Texas A&M University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Approved as to style and content by:

Andreas Klappenecker (Chair of Committee)	Riccardo Bettati (Member)
Krishna Narayanan (Head of Department)	Valerie Taylor (Member)

May 2004

Major Subject: Computer Science

ABSTRACT

Quantum Convolutional Stabilizer Codes. (May 2004)
Neelima Chinthamani, B.S., Jawaharlal Nehru Technological University, India
Chair of Advisory Committee: Dr. Andreas Klappenecker

Quantum error correction codes were introduced as a means to protect quantum information from decoherance and operational errors. Based on their approach to error control, error correcting codes can be divided into two different classes: *block codes* and *convolutional codes*. There has been significant development towards finding quantum block codes, since they were first discovered in 1995. In contrast, quantum convolutional codes remained mainly uninvestigated.

In this thesis, we develop the stabilizer formalism for quantum convolutional codes. We define distance properties of these codes and give a general method for constructing encoding circuits, given a set of generators of the stabilizer of a quantum convolutional stabilizer code, is shown. The resulting encoding circuit enables online encoding of the qubits, i.e., the encoder does not have to wait for the input transmission to end before starting the encoding process. We develop the quantum analogue of the Viterbi algorithm. The quantum Viterbi algorithm (QVA) is a maximum likehood error estimation algorithm, the complexity of which grows linearly with the number of encoded qubits. A variation of the quantum Viterbi algorithm, the Windowed QVA, is also discussed. Using Windowed QVA, we can estimate the most likely error without waiting for the entire received sequence.

To Dad & Mikin

ACKNOWLEDGMENTS

It is a pleasure to thank the people who made this thesis possible.

It is difficult to overstate my gratitude to my graduate advisor, Dr. Andreas Klappenecker. With his enthusiasm, his inspiration, and his great efforts to explain things clearly and simply, he helped to make quantum computing fun for me. Throughout my thesis-writing period, he provided encouragement, sound advice, and lots of good ideas. I would have been lost without him. Thank you so much, Dr. Klappenecker. You are the best ever!

I would like to thank Dr. Bettati and Dr. Narayanan, my committee members, for their interest in this research and for their invaluable suggestions.

Avanti and Santy, for patiently proof reading my thesis.

Thanks to Sumitha and Supriya, for always being there to hear my outrage or glee at the day's events.

Finally, I would like to thank those closest to me. My dad, for creating an environment in which following this path seemed so natural. My guardian angel, mom. Mikin, for his understanding, endless patience and encouragement when it was most required.

This research was supported by NSF grant EIA 0218582 and a Texas A&M TITF initiative.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION TO QUANTUM COMPUTATION

Quantum computing arises from the union of quantum mechanics and classical information theory, and leads to some powerful and surprising possibilities in the quantum context. In this chapter, I first discuss the history of quantum computing. Section B introduces the basics of quantum computing along with the notations, quantum state spaces and quantum gates. In section C, the inherent differences between classical and quantum computers are discussed.

A. Background

Quantum Computers have come a long way since they were first theorized about 20 years ago. Quantum computation is a fundamentally new mode of information processing based on the principles of quantum mechanics. Although, fully functional practical quantum computers have not yet been built, the future of quantum computers seems optimistic.

1. History of Quantum Computing

The idea of a computational device based on quantum mechanics was first explored in the 1970's and early 1980's by physicists and computer scientists such as Charles H. Bennett of the IBM Thomas J. Watson Research Center, Paul A. Benioff of Argonne National Laboratory in Illinois, David Deutsch of the University of Oxford, and Richard P. Feynman of California Institute of Technology. The idea emerged when scientists were pondering the fundamental limits of computation. They understood that if technology continued to abide by Moore's Law, then the continually shrinking size of circuitry packed onto silicon

---

The journal model is *IEEE Transactions on Automatic Control.*

chips would eventually reach a point where individual elements would be no larger than a few atoms. Here a problem arises because at the atomic scale the physical laws that govern the behavior and properties of the circuit are inherently quantum mechanical in nature, not classical.

Feynman was among the first to attempt to provide an answer to this question by producing an abstract model in 1982 that showed how a quantum system could be used to do computations. Later, in 1985, Deutsch realized that Feynman's assertion could eventually lead to a general purpose quantum computer and published a crucial theoretical paper showing that any physical process, in principle, could be modeled perfectly by a quantum computer.

The major breakthrough was achieved by Shor in 1994, when he proposed a method for using quantum computers to crack an important problem in number theory - factorization. He showed how an ensemble of mathematical operations, designed specifically for a quantum computer, could be organized to enable such a machine to factor huge numbers rapidly, much faster than it is thought possible on conventional computers. With this breakthrough, quantum computing transformed from a mere academic curiosity directly into a national and world interest.

B. Basics of Quantum Computation

Among the very popular features of quantum computation are - *quantum parallelism* and *superposition* . Classically, the time it takes to do certain computations can be decreased by using parallel processors. To achieve an exponential decrease in time requires an exponential increase in the number of processors, and hence an exponential increase in the amount of physical space is needed. However, in quantum systems the amount of parallelism increases exponentially with the size of the system. Thus, an exponential

increase in parallelism requires only a linear increase in the amount of physical space needed.

In a quantum computer, the fundamental unit of information is called a quantum bit or qubit. Qubits are represented by systems that have two clearly distinguishable states, $|0\rangle$ and $|1\rangle$. Qubits possess the unique quantum mechanical property known as superposition, i.e., a qubit can simultaneously be in both the one and zero states with a complex coefficient representing the amplitude for each state. The real power of quantum computation derives from the exponential state spaces of multiple quantum bits: just as a single qubit can be in a superposition of 0 and 1, a register of n qubits can be in a superposition of all 2^n possible values. These states that lead to the exponential size of the quantum state space are called the *entangled* states.

A quantum system can undergo two types of operations: quantum state transformations and measurement. In a traditional computer, information is encoded in a series of bits, and these bits are manipulated via Boolean logic gates arranged in succession to produce an end result. Similarly, a quantum computer manipulates qubits by executing a series of quantum gates, each a unitary transformation acting on a single qubit or pair of qubits. On applying these gates in succession, a quantum computer can perform a complicated unitary transformation to a set of qubits that are in some initial input state. The qubits can then be measured, with this measurement serving as the final computational result.

It is still not clear whether the power of quantum parallelism can be harnessed for a wide variety of applications. The factorization algorithm given by Shor and Grover's search algorithm are major examples demonstrating the power of quantum parallelism. The past few years have seen promising small scale prototypes of quantum computers which deal with a small number of qubits. There are a number of proposals for building quantum computers using ion traps, NMR, optical and solid state techniques. All of the

current proposals have scaling problems. The main question is whether useful quantum computers can be built.

The greatest problem for building quantum computers is *decoherence*, the distortion of the quantum state due to interaction with the environment. For some time, it was believed that quantum computers could not be built because it was impossible to isolate them sufficiently from the external environment. The breakthrough came from the invention of quantum error correction techniques. Initially people thought quantum error correction might be impossible because of the impossibility of reliably copying unknown quantum states, but it turns out that this does not prevent the design of quantum error correcting codes that detect certain kinds of errors and enable the reconstruction of the exact error-free quantum state.

1. State Spaces and Bra/Ket Notation

Quantum state spaces can be described in terms of vectors and matrices or in the more popular bra/ket notation. Kets like $|x\rangle$ denote column vectors and are typically used to describe quantum states. The matching bra, $\langle x|$, denotes the conjugate transpose of $|x\rangle$. The basis states $|0\rangle, |1\rangle$ can be expressed as $(1,0)^T, (0,1)^T$. Any superpositions of $|0\rangle$ and $|1\rangle$, $a|0\rangle + b|1\rangle$, can be written as $(a,b)^T \in \mathcal{C}^2$.

$\langle x|y\rangle$ denotes the inner product of the two vectors. For instance, since $|0\rangle$ and $|1\rangle$ are orthogonal we have $\langle 0|1\rangle = 0$. The notation $|x\rangle\langle y|$ is the outer product of $|x\rangle$ and $\langle y|$.

2. Qubits

A qubit is a unit vector in a two dimensional complex vector space for which a particular basis, denoted by $|0\rangle, |1\rangle$, has been fixed. The basic states $|0\rangle$ and $|1\rangle$ are quantum analogues of classical 0 and 1 respectively. Unlike classical bits, qubits can be in a superposition of $|0\rangle$ and $|1\rangle$ such as $a|0\rangle + b|1\rangle$ where a and b are complex numbers such that

$|a|^2 + |b|^2 = 1$. If such a superposition is measured with respect to the basis $|0\rangle, |1\rangle$, then $|0\rangle$ is observed with probability $|a|^2$ and $|1\rangle$ is observed with probability $|b|^2$.

Even though a quantum bit can be put in infinitely many superposition states, it is only possible to extract a single classical bit's worth of information from a single quantum bit. When a qubit is measured, there are only two possible results. Since quantum states cannot be cloned, according to the famous $no - cloning\ theorem$, it is not possible to copy the qubit and measure the copy in a different basis from the original.

3. Quantum Gates

The transformations acting on quantum states must take the original states to transformed states in a way that preserves orthogonality. Any linear transformation on a complex vector space can be described by a matrix and to preserve the orthogonality of the quantum vector spaces these matrices should be unitary. A matrix M is unitary if $MM^* = I$, where M^* is the conjugate transpose of the matrix M. Any unitary transformation of a quantum state space is a legitimate quantum transformation. The fact that quantum transformations are unitary makes them reversible.

The quantum state transformations are also called as *quantum gates*. Some examples of useful single-qubit gates are shown on the following page.

$$\begin{aligned}
\mathcal{I} &= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \\
\sigma_x &= \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \\
\sigma_z &= \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \\
\sigma_y &= i\sigma_x\sigma_z = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}.
\end{aligned}$$

$\mathcal{I}$ is the identity transformation, σ_x is negation, σ_z is a phase shift operation and σ_y is a combination of both. The following are the effect of these transformations on the $\{|0\rangle, |1\rangle\}$ basis. Each of these gates are unitary.

$$\mathcal{I}: \begin{cases} |0\rangle \rightarrow |0\rangle \\ |1\rangle \rightarrow |1\rangle \end{cases} \qquad \sigma_x: \begin{cases} |0\rangle \rightarrow |1\rangle \\ |1\rangle \rightarrow |0\rangle \end{cases}$$

$$\sigma_y: \begin{cases} |0\rangle \rightarrow -|1\rangle \\ |1\rangle \rightarrow |0\rangle \end{cases} \qquad \sigma_z: \begin{cases} |0\rangle \rightarrow |0\rangle \\ |1\rangle \rightarrow -|1\rangle \end{cases}$$

Another important single-bit transformation is the Hadamard Transformation that is defined by

$$H: \begin{cases} |0\rangle \rightarrow \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \\ |1\rangle \rightarrow \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \end{cases}$$

The Hadamard transform has a number of important applications. When applied to $|0\rangle$, the Hadamard gate creates a superposition state $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. Applied to n bits individually, the Hadamard gate generates a superposition of all 2^n possible states, which can be viewed as the binary representation of the numbers from 0 to $2^n - 1$.

The controlled-NOT gate, C_{not}, is a two-qubit gate. It changes the target bit if the control qubit is 1 otherwise leaves it unchanged.

An example of a three qubit gate is the Toffoli gate, which is the controlled-controlled-NOT gate. This gate acts on three qubits two of which act as controls and the third qubit is the target. The gate flips the target qubit only when both the control qubits are 1.

Both Controlled-NOT and Toffoli gates are unitary and thus, reversible.

C. Classical versus Quantum Computers

One of the most noticeable differences between a classical and a quantum computer is that the latter functions at an atomic level, the reason why it is governed by the principles of quantum mechanics. All classical algorithms can be made reversible and can be computed on a quantum computer in comparable time.

Another major difference is that classical computers hardly use error correction. One reason for this is that classical computers use a large number of electrons, so when one goes wrong, the original state can be recovered using the majority logic. A single qubit in a quantum computer is realized using just one or a small number of particles and this creates a need for some sort of error correction. Another reason is that classical computers are digital: after each step, they correct themselves to the closer of 0 or 1. Quantum computers have a continuum of states and they cannot do this without some kind of protection against the errors. Even if the errors introduced are very small, if they are not eliminated, the small errors will build up over the course of the computation, and eventually will derail the computation. Furthermore, quantum states are intrinsically delicate: looking at one collapses it. Hence, error correction is absolutely essential in quantum computers.

Quantum computers are also affected by phase flip errors in addition to bit flip errors as in the case of classical computers. The difference between classical and quantum error correction codes is that, quantum error correction codes should protect the quantum state

from both bit flip and phase flip errors.

CHAPTER II

QUANTUM ERROR CORRECTION

Quantum error correction (QEC) is one of the basic components of quantum information theory. In this chapter, we first explore the reasons that make QEC necessary. We describe in section B the development of quantum error correcting codes. We describe in section C the quantum communication channel and in section D the principles of quantum error correction. The Pauli error model is discussed in section E. We present in section F, Shor's 9-qubit code and discuss in section G stabilizer codes and their properties.

A. Need for Error Correction

It has been proven that quantum information processing can be used to solve problems in cryptography, secure communication and physics simulation exponentially faster than any of its conceivable classical analogues. There are numerous proposals for building quantum computers. These physical models allow exact realizations of quantum information and its manipulation, provided the underlying assumptions are satisfied. However, it is unrealistic to assume that the practical physical systems will behave like the ideal models. Quantum data is very vulnerable to decoherance, interaction with the environment which is due to incomplete isolation of the system from the rest of the world. Also, control errors, which are caused by calibration errors and fluctuations in control parameters, have to be taken care of. Some kind of error correction is necessary to reduce the effects of these errors.

Soon after the existence of quantum error correction was proved in the pioneering paper by Shor [1], the first constructions of good quantum error-correcting codes were given by Steane [2] and Calderbank and Shor [3]. These codes protect the quantum information using additional qubits and make it possible to reverse the effects of the most likely errors.

Encouraged by these positive results, researchers investigated and constructed many new quantum error correcting codes. The fault-tolerant implementations of several quantum operations were also discovered. These implementations make the basic assumption that the effects of all errors are sufficiently small per quantum bit and step of the computation. The requirement on errors is quantified by a maximum tolerable error rate called the *accuracy threshold*. The properties and the assumptions behind the accuracy threshold are given by the *accuracy threshold theorem*. The threshold value depends strongly on the assumed error model and all threshold theorems require that errors at different times and locations be independent. Another requirement states that all basic computational operations can be applied in parallel. Theoretically proven thresholds are not yet practically realizable, but fault-tolerant quantum computation seems feasible. The most attractive aspect of quantum error correction and fault-tolerant computation is that accurate computation does not require perfect physical devices.

B. Background

The first quantum error correcting codes were discovered independently by Shor [1] and Steane [2], as mentioned in the previous section. Shor proved that 9 qubits could be used to protect a single qubit against general errors, while Steane described a general code construction whose simplest example does the same job using 7 qubits. A general theory of quantum error correction dates from subsequent papers of Calderbank and Shor [3] and Steane [4] in which general code constructions, existence proofs, and correction methods were given. Knill and Laflamme [5] and Bennett et. al. [6] provided a more general theoretical framework, describing requirements for quantum error correcting codes, and measures of the fidelity of corrected states. The important concept of the stabilizer (section G) is due to Gottesman [7] and independently Calderbank et. al. [8]; this yielded

many useful insights into the subject, and permitted many new codes to be discovered [7, 8, 9]. Stabilizer methods will probably make a valuable contribution to other areas in quantum information physics. The idea of recursively encoding and encoding again was explored by several authors [10, 11, 12], using quantum resources in a hierarchical way, to permit communication over arbitrarily long times or distances.

Building upon the ideas of quantum error correction, fault-tolerant quantum computation was first proposed by Shor [13]. These ideas were summarized by Preskill [14]. Gottesman put forward a significant number of further ideas on fault-tolerant quantum computing [15], which allow fault tolerant methods to be found for a wide class of QEC codes, and the methods were further improved in [16, 17].

C. Quantum Communication Channel

Communication is the prototypical application of error-correction models. To communicate, a sender needs to transmit information over a noisy channel to the receiver on the other end. During transmission, the information can be affected by disturbances. To protect the information from these errors, the sender encodes it before transmitting it and the receiver decodes this information upon receiving it.

Unlike a classical communication channel, a quantum channel can also be the passage of time during which a set of qubits interacts with its environment, in addition to being a regular communications channel. Also, it can be the result of operating with a noisy gate on some qubits in a quantum computer. In any of these cases, the quantum data needs to be protected against the errors.

D. Principles of Error Correction

The approach used to correct errors involves the following steps:

1. Determine a code, which is a subspace of a larger Hilbert space. The code subspace contains the information to be processed.

2. Identify a decoding procedure that can restore the information represented in the code after an error occurred.

3. Analyze the error behavior of the code and the subsystem.

To protect a given set of quantum data efficiently, the code needs to be capable of limiting errors of the given error model. Also, it needs to have a decoding algorithm which can be implemented easily. The best criteria for selecting the code are the properties of a fixed set of error operators that represent, the most likely errors and then determining whether these errors can be protected against, without any loss of information.

1. Quantum Error Detection

Error detection is used to differentiate which data are correctly received and which ones are in error. If the received information is still in the code subspace it is accepted, else rejected. Given a set of error operators that need to be protected against, the scheme is successful if for each error operator, either the information is unchanged, or the error is detected. An operator E is *detectable* by a code if for each state $|x\rangle$ in the code, either $E|x\rangle = \lambda|x\rangle$ for some scalar λ or $E|x\rangle$ is orthogonal to the code.

Let C be a quantum code, a subspace which represents the quantum information. Let P be an orthogonal projection onto C, and $P^{\perp} = \mathbf{1} - P$ the one that projects onto the orthogonal complement. Then the pair P, $P^{\perp}$ is associated with a measurement that can be used to determine whether a state is in the code or not. If the given state $|x\rangle$, then the result of measurement is $P|x\rangle$ with probability $|P|x\rangle|^2$ and $|P^{\perp}|x\rangle|^2$ otherwise. Using the projection operators, it can be stated that for every state $|x_i\rangle \in C, PE|x_i\rangle = \lambda_E P|x_i\rangle$. Since $P|x\rangle$ is in the code for $|x\rangle$, it follows that $PEP|x\rangle = \lambda_E P|x\rangle$.

The following characterizations can be made for the detectability of errors in quantum states:

1. E is detectable by C if and only if $PEP = \lambda_E P$ for some λ_E.

2. E is detectable by C if and only if for all $|x\rangle, |y\rangle \in C, \langle x|E|y\rangle = \lambda_E \langle x|y\rangle$ for some λ_E.

3. E is detectable by C if and only if for all $|x\rangle, |y\rangle$ in the code with $|x\rangle$ orthogonal to $|y\rangle$, $E|x\rangle$ is orthogonal to $|y\rangle$.

For a given code C, if we can correct errors E and F, then we can correct any linear combination of them, like, $aE + bF$. This linear property implies that we only need to consider whether the code can correct a basis of errors.

A robust error-detecting code should detect as many of the small weight errors as possible. A code C has the *minimum distance* d if the smallest-weight tensor product of Pauli operators E for which C does not detect E is d.

2. Quantum Error Correction

Let $\mathcal{E} = \mathbf{1}, E_1, \ldots$ be a set of error operators. $\mathcal{E}$ is correctable when the errors E_i are unitary operators satisfying the condition that $E_i C$ are mutually orthogonal subspaces. The procedure for decoding is to first make a projective measurement to determine which of the subspaces $E_i C$ the state is in, then apply the inverse of the error operator ${E_i}^\dagger$.

The characterizations for the correctability of a set of error operators $\mathcal{E}$ can be made as:

1. $\mathcal{E}$ is correctable if and only if there is a linear transformation of the set $\mathcal{E}$ such that the operators ${E_i}'$ in the new set satisfy the property the ${E_i}'C$ are mutually orthogonal.

2. $\mathcal{E}$ is correctable if and only if the operators in the set $\mathcal{E}^{\dagger}\mathcal{E} = \{{E_1}^{\dagger}E_2 : E_i \in \mathcal{E}\}$ are detectable.

3. If a code on n qubits has a minimum distance of at least $2e + 1$, then the set of errors of weight at most e is correctable.

E. Error Model

The most investigated error model for qubits is the Pauli error model. The errors are represented using the Pauli spin matrices:

$$\begin{aligned} \mathcal{I} &= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \\ X &= \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \\ Z &= \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \\ Y &= iXZ = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}. \end{aligned}$$

The errors are assumed to occur independently on different qubits. When an error occurs on a qubit, it is equally likely to be a X, Y or Z error. Since, the errors are independent, if an error occurs with a small probability ϵ on a single qubit, the possibility of more than t errors only occurs with a probability $O(\epsilon^{t+1})$.

F. Example: 9-qubit Code

In this example, a single logical qubit is protected against error by encoding it into nine qubits as follows:

$$|\bar{0}\rangle \rightarrow (|000\rangle + |111\rangle)(|000\rangle + |111\rangle)(|000\rangle + |111\rangle) \tag{2.1}$$

$$|\bar{1}\rangle \rightarrow (|000\rangle - |111\rangle)(|000\rangle - |111\rangle)(|000\rangle - |111\rangle) \tag{2.2}$$

This code was first described by P. Shor. Encoding a qubit into 9 qubits spreads the data among nine of them. This code protects the data against one error on a single qubit at a time.

If this data is sent across a quantum channel after encoding, the data will not be affected by the errors under the assumption that the channel affects only one qubit at a time. On the other hand, if the data wasn't encoded, we would have lost the information due to a single error. Even if the data received seems to be correct, we would have no way to verify if this was the data originally sent.

Encoding the data also helps us in finding the errors without actually measuring the qubits, which would destroy the superposition. Suppose a bit flip error occurs on the first qubit of an encoded $|0\rangle$, switching $|0\rangle$ to $|1\rangle$. This error can be located by comparing the first two qubits, which shows us that they are different. By comparing the second and third qubits, we find that they are the same, which tells us that the error has occurred in the first qubit. We then correct it by flipping it back. We use additional qubits, called *ancilla* bits, to find the difference between two qubits. This method can be used to determine the bit flip errors in each of the blocks. Notice however that, this code cannot handle two bit flip errors in a single block. Following the above method, we would flip the third correct bit, thereby introducing more errors.

The quantum data has to deal with phase errors in addition to bit flip errors. A

phase flip affects the qubits in the following way:

$$\begin{aligned} |0\rangle &\rightarrow \quad |0\rangle \\ |1\rangle &\rightarrow -|1\rangle \end{aligned}$$

After a phase flip occurs on the first qubit of the encoded states of the 9-qubit code, the two basis states become

$$|\bar{0}\rangle \rightarrow (|000\rangle - |111\rangle)(|000\rangle + |111\rangle)(|000\rangle + |111\rangle) \tag{2.3}$$

$$|\bar{1}\rangle \rightarrow (|000\rangle + |111\rangle)(|000\rangle - |111\rangle)(|000\rangle - |111\rangle) \tag{2.4}$$

The error can be identified by comparing the first two blocks and later the second two blocks, as in the case of a bit flip error. The difference in this case is that the sign of blocks of three qubits are being compared, whereas earlier individual qubits were compared. Once the block in error is found, it can be fixed by flipping the sign.

Another scenario would be when the bit flip and the phase flip occurs on the same qubit. In such a case, the error can be corrected by first fixing the bit flip and then the phase flip.

All these errors i.e. the bit flip, sign flip and both together can be represented by σ_x, σ_z and σ_y matrices.

The most general error that can occur on a single qubit is an arbitrary 2×2 matrix, but such a matrix can always be written as a linear combination of σ_x, σ_y, σ_z and the 2×2 identity matrix I.

G. Stabilizer Codes

The most popular and useful quantum error correcting codes are based on *stabilizer* constructions. The popularity of stabilizer codes is because of their attractive properties. Using the stabilizer formalism it is easy to determine which Pauli-product errors are

detectable. In addition, they allow to use well-established techniques from the theory of classical error-correcting codes to construct good quantum codes.

A stabilizer of a code encoding k qubits into n qubits is represented using the notation $[[n, k]]$. This code is a 2^k-dimensional subspace C of a 2^n-dimensional Hilbert space that is characterized by the set S of tensor products of Pauli operators that leave each state in the code invariant. More precisely, the code C is the joint eigenspace of the operators in S. The group generated by the stabilizer S is an abelian group that is generated by $(n - k)$ elements.

1. Stabilizer Formalism of Shor's 9-qubit Code

The stabilizer formalism can be explained using the 9-qubit code example introduced in section F. To simplify notation, we use I for identity matrix, $X = \sigma_x, Y = \sigma_y, Z = \sigma_z$. The procedure we used to correct errors for the 9-qubit code was as follows: to detect a bit flip on the first block of three qubits, we compared the first two qubits and then the first and third qubits. This is equivalent to measuring the eigenvalues of Z_1Z_2 and Z_1Z_3. The eigenvalue is +1 if the qubits being compared agree otherwise -1. Similarly, to detect a sign error, we compare the signs of the first and second blocks of three and then the first and the third blocks. This is equivalent to measuring the eigenvalues of $X_1X_2X_3X_4X_5X_6$ and $X_1X_2X_3X_7X_8X_9$. If the signs of the blocks being compared agree then the eigenvalue is +1 else -1.

In order to correct the code, we must measure the eigenvalues of all the eight operators listed in Table I

The two basis states of the 9-qubit code, $|\bar{0}\rangle$ and $|\bar{1}\rangle$, are +1 eigenvectors of all the eight operators given in Table I. These operators form a group S, all the operators in which fix both $|\bar{0}\rangle$ and $|\bar{1}\rangle$. This group S is *stabilizer* of the 9-qubit code. The above eight operators are the generators of this stabilizer.

Table I. Stabilizer for the 9-qubit code

G_1:	Z	Z	I	I	I	I	I	I	I
G_2:	Z	I	Z	I	I	I	I	I	I
G_3:	I	I	I	Z	Z	I	I	I	I
G_4:	I	I	I	Z	I	Z	I	I	I
G_5:	I	I	I	I	I	I	Z	Z	I
G_6:	I	I	I	I	I	I	Z	I	Z
G_7:	X	X	X	X	X	X	I	I	I
G_8:	X	X	X	I	I	I	X	X	X

The errors that can be detected by a particular generator anticommute with that generator. For instance, measuring the eigenvalue of G determines if a bit flip error has occurred on the first qubit or the second, i.e., if X_1 or X_2 has occurred. Both these errors anticommute with G, while the other errors which can be detected such as X_3 through X_9, commute with it. In general, if $G \in S$, and E is an error that can be detected by G, and $|\psi\rangle$ is a valid codeword, then

$$GE|\psi\rangle = -EG|\psi\rangle = -E|\psi\rangle \tag{2.5}$$

so $E|\psi\rangle$ is an -1 eigenvector of G and to detect E we need only measure G.

2. Stabilizer Properties

Let $\mathcal{G}$ denote the group generated by all operators that are tensor products of n matrices in $\{I, X, Y, Z\}$. The stabilizer S is an Abelian subgroup of $\mathcal{G}$ and all the vectors in the coding space $\mathcal{C}$ are fixed by the elements in S. S must be an abelian group, since only commuting operators can have simultaneous eigenvectors.

$\mathcal{G}$ has the following properties:

1. Since $X^2 = Y^2 = Z^2 = +1$, every element in $\mathcal{G}$ squares to $\pm$I.

2. X, Y and Z commute when they are applied on different qubits and anticommute when applied on the same qubit. Therefore, any two elements of $\mathcal{G}$ either commute or anticommute.

3. Every element of $\mathcal{G}$ is unitary, since X, Y and Z are all unitary.

From equation 2.5, we get,

$$\langle\psi_i|E|\psi_j\rangle = \langle\psi_i|ME|\psi_j\rangle = -\langle\psi_i|E|\psi_j\rangle = 0 \tag{2.6}$$

Therefore the code satisfies the following condition,

$$\langle\psi_i|E_a^\dagger E_b|\psi_j\rangle = 0 \tag{2.7}$$

whenever $E = E_a^\dagger E_b = \pm E_a E_b$ anticommutes with G for some $G \in S$, where $E_a^\dagger$ and E_b are error vectors. The above condition should be satisfied by a code, in order to distinguish error E_a acting on one basis codeword ψ_i from error E_b acting on a different basis codeword ψ_j. The errors can be distinguished only when $E_a|\psi_i\rangle$ is orthogonal to $E_b|\psi_j\rangle$.

Since $\langle\psi_i|E|\psi_i\rangle = \langle\psi_j|E|\psi_j\rangle = 0$, if the above condition is satisfied then the code also satisfies,

$$\langle\psi_i|E_a^\dagger E_b|\psi_i\rangle = \langle\psi_j|E_a^\dagger E_b|\psi_j\rangle \tag{2.8}$$

To find what error has occurred, we measure $\langle\psi_i|E_a^\dagger E_b|\psi_i\rangle$ for all possible errors E_a and E_b. This quantity must therefore be the same for all the basis codewords. This condition is captured in 2.8.

If $E_a^\dagger E_b$ anticommutes with some element of S for all errors E_a and E_b in some set, the code corrects all the error vectors spanned by that set.

In conclusion, we can say that a quantum code with stabilizer S will detect all errors E that are either in S or anticommute with some element of S.

CHAPTER III

QUANTUM CONVOLUTIONAL CODING

Error correcting codes can be divided into two different classes based on their approach to error control : *block codes* and *convolutional codes*.

In section A we discuss the differences between block codes and convolutional codes. In section B, the classical convolutional encoders and the distance properties of the classical convolutional codes are described.

In section C, we introduce quantum convolutional stabilizer codes and discuss their distance properties. Section D contains the description of an example of a quantum convolutional stabilizer code.

A. Block Codes versus Convolutional Codes

In a block code case, the original state is first divided into finite blocks of same length. Each block is then encoded separately and the encoding is independent of the state of the other blocks. The code is decoded by first measuring the error syndromes of the encoded blocks, and then applying a necessary unitary transformation to the corresponding erroneous qubits. The unitary transformations are independent of the ones applied to the other blocks. Since there is only a finite number of error syndromes and hence only a finite number of recovery operations per block, decoding a quantum block code requires only finite number of operations.

Besides block codes, convolutional codes are well known in classical error correction. Convolutional codes can be viewed either as nonblock linear codes over a finite field or as block codes over certain infinite fields. The choice between block and convolutional codes depends on the application for which they are being used. Block codes are particularly

powerful when the channel has *burst errors*, which is the case in secondary memories such as magnetic tapes and disks.

Unlike as in block codes, the encoding operation in convolutional codes depends on the current as well as a number of past information bits. Convolutional codes are aimed at protecting a stream of information being transmitted continuously by converting it to a single codeword regardless of its length. The most attractive attribute of the convolutional codes is their decoding algorithm. They have a *maximum likelihood* error estimation algorithm for all memoryless channels with a complexity growing linearly with the number of encoded qubits as compared to a generic family of block codes, for which the maximum likelihood error estimation algorithm has in general a complexity growing exponentially with the number of encoded qubits.

Another advantage of convolutional codes is their ability to perform encoding and decoding of the bits online. It is not necessary to wait for all the bits before starting the encoding or the decoding process.

B. Classical Convolutional Codes

Convolutional codes are one of the most widely used error correcting codes in practical communication systems. These codes are developed with a strong mathematical structure and are primarily used for real time error correction. Convolutional codes convert the entire data stream into one single codeword. The n encoder outputs at any given time unit depend not only on the current k input bits but also on m previous input blocks. An (n, k, m) convolutional code can be implemented with a k-input, n-input linear sequential circuit with input memory m. Typically, n and k are small integers but the memory order m must be made large to achieve low error probabilities.

Convolutional codes were first introduced by Elias in 1955 as an alternative to block

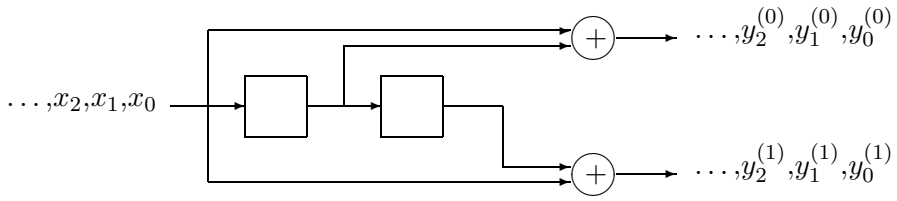

Fig. 1. A Rate-1/2 Classical Convolutional Encoder

codes. Many approaches to decoding convolutional codes were derived by the following decade. The most popular decoding algorithm so far is the maximum likelihood decoding algorithm proposed by Viterbi in 1967. This was relatively easy to implement for codes with small memory orders. This scheme, called Viterbi decoding, led to the application of convolutional codes to deep-space and satellite communication in the early 1970s.

1. Classical Convolutional Encoders

The encoder for a binary $(2, 1, 2)$ code is shown in Fig. 1. The code generated by this generator has a rate-1/2, which is established by the fact that the encoder outputs two bits for every input bit. In general, an encoder with k inputs and n outputs is said to have rate k/n. The encoder consists of an $m = 2$-stage shift register together with n=2 modulo-2 adders and a multiplexer for serializing the encoder outputs. Since mod-2 addition is a linear operation, the encoder is a linear feed forward shift register. All convolutional encoders can be implemented using a linear feed forward shift register of this type.

The input sequence $\mathbf{x} = (x_0, x_1, x_2, \ldots)$ enters the shift-register circuit consisting of a series of memory elements. The encoder outputs two output sequences $y^{(0)} = (y_0^{(0)}, y_1^{(0)}, y_2^{(0)}, \ldots)$ and $y^{(1)} = (y_0^{(0)}, y_1^{(0)}, y_2^{(0)}, \ldots)$ that are the convolution of the input sequence $\mathbf{x}$ and the contents of the m memory elements. These encoded output sequences can be multiplexed to create a single coded data stream $\mathbf{y} = (y_0^{(0)}, y_0^{(1)}, y_1^{(0)} y_1^{(1)}, y_2^{(0)}, y_2^{(1)}, \ldots)$. $\mathbf{y}$ is the convolutional codeword.

For linear convolutional encoders, if y_1 and y_2 are the codewords corresponding to inputs x_1 and x_2, respectively, then (y_1+y_2) is the codeword corresponding to the input (x_1+x_2).

An *impulse response* $g_j^{(i)}$ is obtained for the ith output of an encoder by applying a single 1 at the jth input followed by a string of zeros. Strings of zeros are applied to all other inputs. The impulse responses for the encoder in Fig. 1 are

$$\begin{aligned} g^{(0)} &= (110) \\ g^{(1)} &= (101) \end{aligned} \tag{3.1}$$

The impulse responses are often referred to as generator sequences, because their relationship to the codewords generated by the corresponding convolutional encoder is similar to that between generator polynomials and codewords in a cyclic code. The generator sequences have been terminated at the point beyond which all of the output streams contain nothing but zeros. Since there are two memory elements in the shift register in Fig. 1, each bit in the input data stream can affect at most 3 bits, which is the same as the length of the generator sequences. In general, the amount of memory in the encoder determines the extent to which an input bit directly affects the output data streams.

Using the generator sequences described above, the output sequences of an encoder can be given using the following general form.

$$y_i^{(j)} = \sum_{l=0}^{m} x_{i-l} g_l^{(j)} \tag{3.2}$$

Each coded output sequence $y^{(j)}$ in a rate-$1/n$ code is the convolution of the input sequence $\mathbf{x}$ and the impulse response $g^{(i)}$.

$$\mathbf{y}^{(j)} = \mathbf{x} * \mathbf{g}^{(j)} \tag{3.3}$$

Equation 3.3 can be generalized for a k-input encoder as,

$$\mathbf{y}^{(j)} = \sum_{t=0}^{k-1} (\mathbf{x}^{(t)} * \mathbf{g}_t^{(j)}) \tag{3.4}$$

Equation 3.2 can be reexpressed as a matrix multiplication operation, thus providing a generator matrix similar to that developed for block codes. The primary difference arises from the fact that the input sequence is not necessarily bounded in length, and thus the generator and parity check matrices for classical convolutional codes are semi-infinite.

The generator matrix for a rate-1/2 code is formed by interleaving the two generator sequences $\mathbf{g}^{(0)}$ and $\mathbf{g}^{(0)}$ as follows,

$$\mathbf{G} = \begin{pmatrix} g_0^{(0)}g_0^{(1)} & g_1^{(0)}g_1^{(1)} & g_2^{(0)}g_2^{(1)} & \cdots & g_m^{(0)}g_m^{(1)} & & 0 \\ & g_0^{(0)}g_0^{(1)} & g_1^{(0)}g_1^{(1)} & g_2^{(0)}g_2^{(1)} & \cdots & g_m^{(0)}g_m^{(1)} & \\ & & g_0^{(0)}g_0^{(1)} & g_1^{(0)}g_1^{(1)} & g_2^{(0)}g_2^{(1)} & \cdots & g_m^{(0)}g_m^{(1)} \\ 0 & & \ddots & \ddots & \ddots & \cdots & \ddots \end{pmatrix} \tag{3.5}$$

The code word $\mathbf{y}$ corresponding to an input sequence $\mathbf{x}$ is then obtained through matrix multiplication.

$$\mathbf{y} = \mathbf{x}\mathbf{G} \tag{3.6}$$

2. Distance Properties of Classical Convolutional Codes

The performance of a convolutional code depends on the decoding algorithm employed and the distance properties of the code. Several distance measures are introduced here.

The most important distance measure for convolutional codes is the *minimum free distance* d_{free}, defined as

$$d_{free} = min\{d(\mathbf{y}', \mathbf{y}'' : \mathbf{x}' \neq \mathbf{x}''\} \tag{3.7}$$

where $\mathbf{y}'$ and $\mathbf{y}''$ are the code words corresponding to the input sequences $\mathbf{x}'$ and $\mathbf{x}''$,

respectively. It is assumed that if $\mathbf{x}'$ and $\mathbf{x}''$ are of different lengths, zeros are added to the shorter sequence so that their corresponding code words have equal lengths. Hence, d_{free} is the minimum distance between any two code words in the code. Since a convolutional code is a linear code,

$$\begin{aligned} d_{free} &= min\{w(\mathbf{y}' + \mathbf{y}'') : \mathbf{x}' \neq \mathbf{x}''\} \\ &= min\{w(\mathbf{y}) : \mathbf{x} \neq \mathbf{0}\} \\ &= min\{w(\mathbf{xG}) : \mathbf{x} \neq \mathbf{0}\} \end{aligned}$$

where $\mathbf{y}$ is the code word corresponding to the input sequence $\mathbf{x}$. Hence, d_{free} is the minimum-weight code word of any length produced by a nonzero input sequence.

Another important distance measure for convolutional codes is the *column distance function* (CDF). Let $[\mathbf{x}]_i$ and $[\mathbf{y}]_i$ denote a input sequence $\mathbf{x}$ and a code word $\mathbf{y}$ truncated after ith block, respectively. They are given as,

$$[\mathbf{x}]_i = (x_0^{(1)} x_0^{(2)} \cdots x_0^{(k)}, x_1^{(1)} x_1^{(2)} \cdots x_1^{(k)}, \ldots, x_i^{(1)} x_i^{(2)} \cdots x_i^{(k)})$$

$$[\mathbf{y}]_i = (y_0^{(1)} y_0^{(2)} \cdots y_0^{(n)}, y_1^{(1)} y_1^{(2)} \cdots y_1^{(n)}, \ldots, y_i^{(1)} y_i^{(2)} \cdots y_i^{(n)})$$

The column distance function of order i, d_i, is defined as

$$\begin{aligned} d_i &= min\{d([\mathbf{y}']_i + [\mathbf{y}'']_i) : [\mathbf{x}']_0 \neq [\mathbf{x}'']_0\} \\ &= min\{w([\mathbf{y}]_i) : [\mathbf{x}]_0 \neq \mathbf{0}\} \end{aligned}$$

where $\mathbf{y}$ is the code word corresponding to the input sequence $\mathbf{x}$. Hence, d_i is the minimum-weight code word over the first $(i + 1)$ time units whose initial input block

is nonzero. In terms of the generator matrix of the code,

$$[\mathbf{y}]_i = [\mathbf{x}]_i[\mathbf{G}]_i \tag{3.8}$$

where $[\mathbf{G}]_i$ is a $k(i+1) \times n(i+1)$ submatrix of $\mathbf{G}$. Then,

$$d_i = min\{w([\mathbf{x}]_i[\mathbf{G}]_i) : [\mathbf{x}]_0 \neq \mathbf{0}\} \tag{3.9}$$

For the special case, when $i = m$, d_m is called the *minimum distance* of convolutional code and is also denoted as d_{min}. It represents the minimum-weight code word over the first constraint length whose initial input block is nonzero.

C. Quantum Convolutional Codes

1. Background

The theory of quantum convolutional codes has not been developed significantly so far. Quantum convolutional codes were first introduced in 1997 by Chau [18, 19]. In [18], Chau reported two general methods to construct QCCs. Both the methods involve encoding twice. The first method is to encode with a classical linear convolutional code and then applying a quantum block code. The second method is to encode using two classical convolutional codes, one that corrects bit flip errors and the other that corrects phase errors. The claim is made that if the first code can correct bit flip errors and the second can correct phase flip errors, the resulting code corrects both bit flip and phase flip errors. This generalization was shown to be in error by Knill. Although, the claim applies to the example described in this paper, it is false in general. In [19], Chau also stated the conditions necessary for the existence of a QCC whose decoding error does not propagate indefinitely were.

More recently, Ollivier and Tillich described a rate 1/5 quantum convolutional code

[20] that uses the stabilizer formalism. A general consistent theory of quantum convolutional codes had not been developed so far.

2. Quantum Convolutional Stabilizer Codes

Let $\mathcal{H}_2^\infty = \bigotimes_{i=1}^{\infty} \mathcal{C}^2$ denote the ambient space. Let $\mathcal{G}$ denote the group generated by operators of form $\bigotimes_{i=1}^{\infty} e_i$, where $e_i \in \{I, X, Y, Z\}$ and almost all e_i are equal to I. The *Quantum Convolutional Stabilizer (QCS) code* is a subspace of an infinite dimensional Hilbert space $\mathcal{H}_2^\infty$, that is given as a joint $+1$-eigenspace of an abelian subgroup of $\mathcal{G}$.

The Stabilizer for a general QCS code is given below. Each of the g_i^j represents a Pauli spin matrix. The length of each of the generators and hence the the dimension of the stabilizer is semi-infinite and they depend on the length of the input sequence that needs to be encoded. The code subspace is given by the common $+1$-eigenspace of the generators in the stabilizer of the code. All the generators commute and are linearly independent, i.e., no generator can expressed as product of other generators.

$$\begin{aligned}
M_1 &= g_1^1 \otimes g_1^2 \otimes g_1^3 \otimes g_1^4 \otimes g_1^5 \cdots \\
M_2 &= g_2^1 \otimes g_2^2 \otimes g_2^3 \otimes g_2^4 \otimes g_2^5 \cdots \\
M_3 &= g_3^1 \otimes g_3^2 \otimes g_3^3 \otimes g_3^4 \otimes g_3^5 \cdots \\
&\vdots \\
M_d &= g_d^1 \otimes g_d^2 \otimes g_d^3 \otimes g_d^4 \otimes g_d^5 \cdots \\
M_{ni+j} &= I^{\otimes ni} \otimes M_j, \quad 0 < i, 1 \leq j \leq d
\end{aligned}$$

A *rate-k/n* QCS code transforms every block of k input qubits into a superposition of states containing n output qubits. This transformation depends on the m qubits of the preceding and the following blocks. The integer m is called the *block overlap* of the code.

The QCS code with a rate k/n contains d original generators in its *stabilizer*, where $d = n - k$. The remaining generators in the stabilizer are the shifted versions of these d

generators, shifted to the right by multiples of n. A block of d generators act on $(n+m)$ consecutive qubits. Such consecutive blocks overlap. The *constraint length* of the code, denoted by K, is defined as the number of blocks that overlap with any particular block.

Though in principle, a QCS code contains codewords of infinite length, for practical purposes we consider codewords whose input vectors are truncated after i input blocks. This is justified because the input transmission stops at some point of time. A rate k/n QCS code would contain k qubits in each input block. The encoded codewords of such truncated input sequences will have $ni + m$ qubits, where m is the block overlap. We denote the QCS code that contains codewords generated by truncated input sequences using $\mathcal{Q}_i$.

Along with the generators given above, we have, at most $[(n-k)K]/2$ *initializing* and at most $[(n-k)K]/2$ *terminating* generators that are applied to the initial and the trailing qubits, respectively. The integer K is the constraint length of the code. Out of each of these $[(n-k)K]/2$ generators, we consider only those generators that contain non-identity Pauli matrices. The initializing generators, denoted by the set $\{M_0\}$, can be obtained by capturing the overlap between the original generator block and the blocks preceding it (these blocks can be viewed as original d generators shifted to the left by multiples of n). Similarly, the set of terminating generators, denoted by the set $\{M_\infty\}$, can be obtained by capturing the overlap in the generator block following the final block of encoding generators.

For two distinguishable states, such that $\langle\psi_1|\psi_2\rangle = 0$, the *quantum Hamming distance* between the two states is given by qd, if all errors of weight up to qd applied on the states $|\psi_1\rangle$ and $|\psi_2\rangle$, result in two states that are still distinguishable, i.e., $\langle\psi_1|E|\psi_2\rangle = 0$ for all errors E of weight up to qd.

Similar to the their classical analogues, we can define the *Column Distance Function*, *Minimum Distance* and *Minimum Free Distance* for the quantum convolutional codes.

Table II. Original generators for a rate-1/5 quantum convolutional code

M_1:	X	Z	X	I	I	I	I	I	...
M_2:	I	I	I	Z	X	Z	X	I	...
M_3:	Y	Y	X	X	Z	I	I	I	...
M_4:	X	Z	I	I	I	Z	X	I	...

The *quantum column distance function*(QCDF) qd_i of a rate k/n quantum convolutional code is the minimum quantum Hamming distance between all pairs of distinguishable codewords of the corresponding truncated code $\mathcal{Q}_i$.

$$qd_i \equiv \min\{qd(|\psi_i^{'}\rangle, |\psi_i^{"}\rangle) \mid \langle\psi_i^{'}|\psi_i^{"}\rangle = 0\} \quad \text{where} |\psi_i^{'}\rangle, |\psi_i^{"}\rangle \in \mathcal{Q}_i \tag{3.10}$$

The *minimum distance* qd_{min} of an (n, k) quantum convolutional code with constraint length K is the QCDF qd_i evaluated at $i = (K + 1)$.

The *minimum free distance*, denoted by qd_{free}, is the minimum Hamming distance between all pairs of complete convolutional code words.

$$qd_{free} \equiv \min\{qd(|\psi^{'}\rangle, |\psi^{"}\rangle) \mid \langle\psi^{'}|\psi^{"}\rangle = 0\} \tag{3.11}$$

D. Example

Let us have a look at a simple example. We have a rate-1/5 code, the first block of 4 original generators of which are given in Table II. The remaining generator block of the stabilizer can be obtained by shifting these generators to the left by multiples of 5, that is,

$$M_{5i+j} = I^{\otimes 5i} \otimes M_j, \quad 0 < i, 1 \leq j \leq 4 \tag{3.12}$$

The generators in each block act on 7 consecutive qubits and they commute and are

linearly independent. Also, the generators in a particular block affect the first two qubits of the following block and the last two qubits of the previous block. The block overlap of the code is $m = 2$. The constraint length K of this code is 2 because every generator block overlaps with two other blocks, the one preceding it and one following it.

As mentioned in the previous section, we can have up to $[(n-k)K]/2$ distinct initializing and up to $[(n-k)K]/2$ distinct terminating generators acting on the preceding and the trailing qubits. For our code, we have one non-trivial initializing generator denoted by M_0^1. We also have two terminating generators denoted by M_∞^1 and M_∞^2. The initializing and terminating generators are given by:

$$\begin{array}{lcccccc} M_0^1 & : & Z & X & I & I & I & \dots \\ M_\infty^1 & : & \dots & I & I & I & X & Z \\ M_\infty^2 & : & \dots & I & I & I & Y & Y \end{array}$$

CHAPTER IV

ENCODING AND DECODING QUANTUM CONVOLUTIONAL STABILIZER CODES

One of the major advantages of convolutional codes is their ability to perform online encoding and decoding. This chapter discusses the encoding and decoding techniques used for quantum convolutional stabilizer codes introduced in the previous chapter.

In Section A, the construction of an encoding circuit, given the original generator block of the stabilizer of the QCS code, is described with the help of an example.

In Section B, the quantum analogue of the Viterbi algorithm is given. Also, a variation of it, the windowed Viterbi algorithm, is discussed.

A. Constructing Encoding Circuits

The special structure of the stabilizer of a QCS code can be exploited to construct a quantum circuit that enables us to encode qubits online. We next describe the projection operator used to compute codewords and the procedure to implement this projection operator using a quantum gate array.

1. Projection Operator for Encoding

To map our input sequence to the code subspace, we apply a projection operator P. As mentioned before, even though our code subspace contains vectors of infinite length, in practice we need only handle vectors of finite length since the input transmission stops after some amount of time.

Let $\mathcal{Q}_i$ be our QCS code, the code words of which are truncated after i output blocks.

The projection operator to encode using $\mathcal{Q}_i$ is given by,

$$\begin{aligned} P_i = \quad & [\frac{1}{\sqrt{2^l}} \prod_{p=1}^{l} (I + M_0^p)] \times \\ & [\frac{1}{\sqrt{2^{(n-k)i}}} \prod_{r=0}^{i-1} (I + M_{nr+1})(I + M_{nr+2}) \ldots (I + M_{nr+d})] \times \\ & [\frac{1}{\sqrt{2^t}} \prod_{q=1}^{t} (I + M_\infty^q)] \end{aligned} \tag{4.1}$$

where $d = (n - k)$. l is the number of initializing generators denoted by the set $\{M_0\}$, t is the number of terminating generators denoted by the set $\{M_\infty\}$.

Before applying the projection operator we first add some ancilla qubits to the input stream. The k input qubits in each input block are placed such that qubit q_i^j is at position $(m + (n - 1)i + j)$ where m is the block overlap of the code, i is the index of the input block and j is the index of qubit within an input block. The encoding is given by,

$$\begin{aligned} & |q_1^1, \cdots, q_1^k\rangle \otimes \ldots \otimes |q_i^1, \cdots, q_i^k\rangle \longmapsto \\ & P_i[|\overbrace{0, \ldots, 0}^{m}\rangle \otimes |\overbrace{0, \ldots, 0}^{n-k}, q_1^1, \cdots, q_1^k\rangle \otimes \ldots \otimes |\overbrace{0, \ldots, 0}^{n-k}, q_i^1, \cdots, q_i^k\rangle] \end{aligned} \tag{4.2}$$

Our truncated QCS code $\mathcal{Q}_i$ is a unitary mapping of $\mathcal{H}_2^{ik}$ into $\mathcal{H}_2^{ni+m}$. Strictly speaking, this is actually an isometric mapping of $\mathcal{H}_2^{ik}$ into a 2^{ik}-dimensional subspace of $\mathcal{H}_2^{ni+m}$. The input sequences range over all vectors in $\mathcal{H}_2^{ik}$ and they form distinct basis states. And each of these input sequences is associated with a quantum codeword given by the mapping in equation 4.2. These 2^{ik} basic codewords are all valid quantum states and are mutually orthogonal since they are unitarily transformed from 2^{ik} distinct basic input states.

Also, all the codewords are fixed by the stabilizer, i.e., they lie within the specified code. This is because for each generator M_i, the following equation holds:

$$M_i \cdot (I + M_i) = (M_i + I) \tag{4.3}$$

Hence, applying a generator on the codeword leaves it unchanged.

2. Construction of Encoding Circuits

Before we proceed to describe the algorithm to construct an encoding circuit, we introduce some useful vector notations. The X-vector of a generator M_i denoted as X_{M_i} is given by,

$$(X_{M_i})_j = \begin{cases} 1 & \text{if } M_{ij} = X \text{ or } Y; \\ 0 & \text{if } M_{ij} = I \text{ or } Z. \end{cases} \tag{4.4}$$

The Z-vector of a generator M_i denoted as Z_{M_i} is given by,

$$(Z_{M_i})_j = \begin{cases} 1 & \text{if } M_{ij} = Z \text{ or } Y; \\ 0 & \text{if } M_{ij} = I \text{ or } X. \end{cases} \tag{4.5}$$

The *X-matrix* of generators $M_1, \ldots, M_d$ is a $(n+m) \times d$ matrix, denoted as X_M, where

$$(X_M)_{ji} = \begin{cases} 1 & \text{if } M_{ij} = X \text{ or } Y; \\ 0 & \text{if } M_{ij} = I \text{ or } Z. \end{cases} \tag{4.6}$$

The *Z-matrix* of the generator is similarly defined.

To obtain an encoding circuit, we have to first transform the X-matrix into the *pseudo-standard form*. A $p \times q$ matrix is said to be in the pseudo-standard form, if it contains an identity submatrix of order $q \times q$. The rows of this identity matrix do not necessarily have to be in order.

Following is procedure used to construct the gate array that computes the mapping given in equation 4.2.

1. *Convert the original generator block into pseudo-standard form:*

   (a) Truncate the vectors in the original generator block to length $(n+m)$.

   (b) Obtain the X- and Z-matrices of the original generator block.

(c) Perform corresponding column transformations on X- and Z-matrices until we have an identity submatrix of order $(n-k)$ in the X-matrix. The rows of this submatrix do not necessarily have to be in order.

2. *Mark the positions of $(n-k)$ ancillary qubits, m memory qubits and k input qubits in the quantum circuit:*

   (a) Identify the rows in the X-matrix which belong to the identity submatrix. The diagonal elements of the identity matrix mark the positions of $(n-k)$ ancillary qubits in the encoding circuit.

   (b) Pick any m non-zero rows in the X-matrix. Mark these as the positions of the memory ancilla qubits.

   (c) Mark the remaining k rows as the positions of the input qubits.

3. *Construct the encoding circuit:*

   (a) Apply Hadamard transform on each of the $(n-k)$ ancilla qubits

   (b) Apply each of the $(n-k)$ generators on the block of $(n+m)$ qubits, whose positions are determined from Step 2. These generators are conditioned on each of the $(n-k)$ ancillary qubits identified in Step 2-(a).

   (c) Apply the same gate array shifted down n qubits and repeat until the end of the input sequence

4. *Apply initializing and terminating generators*

   (a) Obtain the transformed initializing and terminating generators from the transformed generator block.

   (b) Apply the initializing generators on the initial m qubits and terminating generators on the m trailing qubits of the last encoded block.

It is helpful to consider an example to understand the above algorithm. Let us take the example given in the Section D of the previous chapter. The first step is to truncate the set of original generators to length $(n+m)$. This has trivial effect on the generators, since the truncated portion of the generators contains only identity operators.

For our example, the $d(=(n-k)=(5-1)=4)$ original generators are given as,

$$\begin{array}{lccccccc} M_1\text{:} & \mathrm{X} & \mathrm{Z} & \mathrm{X} & \mathrm{I} & \mathrm{I} & \mathrm{I} & \mathrm{I} \\ M_2\text{:} & \mathrm{I} & \mathrm{I} & \mathrm{I} & \mathrm{Z} & \mathrm{X} & \mathrm{Z} & \mathrm{X} \\ M_3\text{:} & \mathrm{Y} & \mathrm{Y} & \mathrm{X} & \mathrm{X} & \mathrm{Z} & \mathrm{I} & \mathrm{I} \\ M_4\text{:} & \mathrm{X} & \mathrm{Z} & \mathrm{I} & \mathrm{I} & \mathrm{I} & \mathrm{Z} & \mathrm{X} \end{array}$$

For the second step, we take all the generators in the block and split them into two vectors: *X-vector* and *Z-vector*. The X- an d Z-matrices of our generator block are,

$$X_M = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \qquad Z_M = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

As can be seen, the X-matrix does not contain a 4×4 identity matrix. We perform corresponding column transformations on both the X- and Z-matrices until we obtain a 4×4 order identity matrix in the X-matrix. We allow only column transformations because the only transformations that do not affect the characteristics of the code are those that change a generator M_i into $M_i \cdot M_j$ where $j \neq i$. In terms of matrices X_M and Z_M, such an operation adds the jth column to the ith column in both matrices. Another transformation that does not affect the code characteristics is the one in which the n qubit

positions are reordered, which corresponds to a reordering of the rows in both matrices. However, this is not a valid transformation for the convolutional case because when we reorder the qubits, the generators in overlapping blocks may no longer commute.

We apply the following two column transformation on both X- and Z-matrices.

$$C_2 \mapsto C_2 + C_4$$

$$C_3 \mapsto C_3 + C_1$$

The transformed X- and Z-matrices are

$$X'_M = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad Z'_M = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

The rows $3, 5, 4, 7$ of X'_M form a 4×4 identity matrix. We mark the positions of the 4 ancillary qubits using these rows as follows,

$$a_1 \mapsto 3$$

$$a_2 \mapsto 5$$

$$a_3 \mapsto 4$$

$$a_4 \mapsto 7$$

As per step 2-(b) of the algorithm, we pick rows 1,2 as the positions of the 2 memory qubits, m_1 and m_2. The remaining row 6 corresponds to the input qubit q.

The truncated generator matrix corresponding to the matrices $X_M^{'}$ and $Z_M^{'}$ is

$$[M^{'}]_7 = \begin{pmatrix} X & Z & X & I & I & I & I \\ X & Z & I & Z & X & I & I \\ Z & X & I & X & Z & I & I \\ X & Z & I & I & I & Z & X \end{pmatrix}$$

Now that we determined the positions of all the necessary qubits for each encoded block, we can translate the X- and Z-matrices into our gate array.

Our intention is to obtain the following map

$$\begin{pmatrix} m_1 \\ m_2 \\ a_1 \\ a_3 \\ a_2 \\ q \\ a_4 \end{pmatrix} \quad \mapsto \quad [M^{'}]_7^T \begin{pmatrix} m_1 \\ m_2 \\ a_1 \\ a_3 \\ a_2 \\ q \\ a_4 \end{pmatrix} \tag{4.7}$$

We split the mapping given in equation 4.7 into two parts to better understand the gate array construction,

$$\begin{pmatrix} m_1 \\ m_2 \\ a_1 \\ a_3 \\ a_2 \\ q \\ a_4 \end{pmatrix} \quad \mapsto \quad X_M^{'} \begin{pmatrix} m_1 \\ m_2 \\ a_1 \\ a_3 \\ a_2 \\ q \\ a_4 \end{pmatrix} \quad \mapsto \quad Z_M^{'} \begin{pmatrix} m_1 \\ m_2 \\ a_1 \\ a_3 \\ a_2 \\ q \\ a_4 \end{pmatrix} \tag{4.8}$$

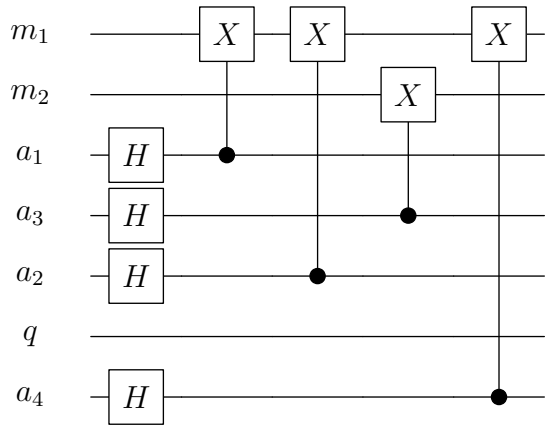

Fig. 2. Gate Array Corresponding to the X-matrix

The first part of the mapping in equation 4.8 is computed by the gate array shown in Fig. 2.

In Fig. 2, we can see how the operations in the gate array correspond to the entries of the matrix X'_M. What we have done is to apply each column conditioned on one of its elements. These elements correspond to the diagonal elements of the 4×4 identify submatrix of the matrix X'_M. The 4 ancilla bits act as the controls for each of the 4 columns. We can only do this if the control qubits have not been changed before we get around to applying that generator. If any earlier column had a 1 in the same row as the ancilla bits, then that qubit might have been changed before it could be used. This is why we need to put the X-matrix is pseudo-standard form - each column of the matrix must be conditioned on the input of the corresponding ancilla qubit, so the diagonal elements must be first in their rows. A matrix in our pseudo-standard form has this property, while a more general matrix does not.

We now implement the second part of mapping in equation 4.8 by including the phase shifts. We insert conditional phase gates into Fig. 2 in accordance with the matrix Z'_M

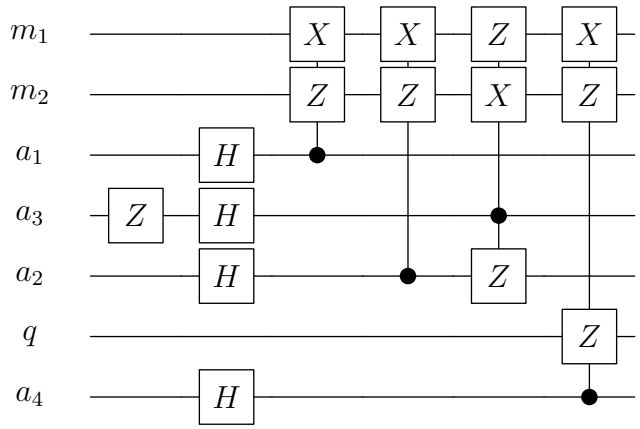

Fig. 3. Gate Array Corresponding to a Generator Block

as shown in Fig. 3.

Note that the Z operator on ancilla qubit a_3 is applied before the Hadamard gate. This is because the Hadamard gates are always applied right before the controls. And since the Z operator does not have any effect on $|0\rangle$, it can be neglected.

We apply shifted version of this entire gate array to the succeeding blocks of qubits. These shifted gate arrays correspond to the succeeding generator blocks. We repeat this process until we encode all the qubits in the input sequence.

Using the transformed generator matrix $[M^{'}]_7$, we reconstruct our stabilizer as,

$$\begin{aligned}
M_1: &\quad X \quad Z \quad X \quad I \quad I \quad I \quad I \quad I \quad \dots \\
M_2: &\quad X \quad Z \quad I \quad Z \quad X \quad I \quad I \quad I \quad \dots \\
M_3: &\quad Z \quad X \quad I \quad X \quad Z \quad I \quad I \quad I \quad \dots \\
M_4: &\quad X \quad Z \quad I \quad I \quad I \quad Z \quad X \quad I \quad \dots
\end{aligned}$$

$$M_{5i+j} = I^{\otimes 5i} \otimes M_j, \quad 0 < i, 1 \leq j \leq 4$$

We now have one initializing generator and two terminating generators, given by:

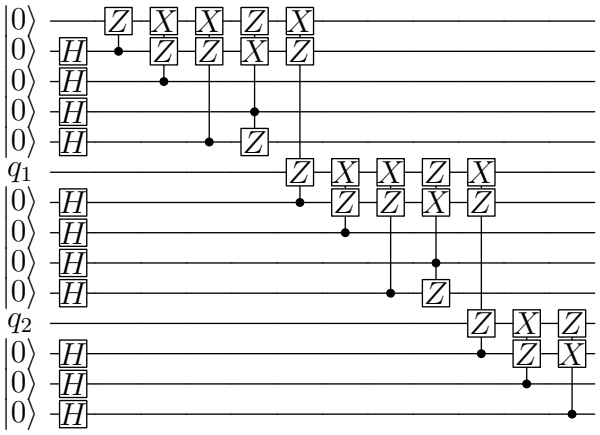

Fig. 4. Encoding Circuit for Rate-1/5 QCS Code

$$
\begin{aligned}
M_0^1 &: \; Z \quad X \quad I \quad I \quad I \quad \ldots \\
M_\infty^1 &: \; \ldots \quad I \quad I \quad I \quad X \quad Z \\
M_\infty^3 &: \; \ldots \quad I \quad I \quad I \quad Z \quad X
\end{aligned}
$$

The circuit is completed by applying the gates corresponding to the initializing generator to the initial m memory qubits in the circuit and applying the terminating generators to the last m qubits in the circuit.

This standard form corresponds to the case where the matrix X_M has a full rank. The complete encoding circuit is given in Fig. 4.

B. Decoding Quantum Convolutional Codes

The final stage in the error correcting process is decoding the transmitted sequence. The input sequence is protected against noise by encoding it using an error correcting code before transmitting it through a channel, which might introduce errors in it. During the decoding process, we try to figure out if the received sequence is the same as the one

that was transmitted or if has any errors in it. Then we try to estimate what, if any, error has occurred. Once the errors are found, we undo them by applying appropriate transformations. After the received sequence is fixed, we try to retrieve the original input sequence. There is always a chance that the error we estimate is not the one that has actually occurred. In such a case, when we try to fix the error, instead of correcting the error we add more errors. This is can be avoided to some extent by selecting an error correcting code with good error correcting capabilities or by picking a efficient error estimation algorithm. The efficiency of the error estimation algorithm depends on its capability to estimate the most likely error that has occurred.

For classical convolutional codes, Viterbi algorithm provides both a maximum-likelihood and a maximum a posteriori error estimation, the complexity of which increases linearly with the number of qubits in the received sequence. In this section the quantum analogue of the classical Viterbi algorithm is given.

Once we have the received sequence, the first step is to determine if it is correct or has errors in it. We proceed to do this using measuring the error syndromes as described in [21]. Let $f_M : \mathcal{G} \to Z_2$

$$f_M(E) = \begin{cases} 0 & \text{if } [M, E] = 0; \\ 1 & \text{if } M, E = 0. \end{cases} \tag{4.9}$$

and $f(E) = (f_{M_0}(E), f_{M_1}(E), f_{M_2}(E), \ldots)$, where $M_0, M_1, M_2, \ldots$ are generators of S. Then $f(E)$ is some sequence of bits which is 0 if and only if E commutes with all the generators $M_0, M_1, M_2, \ldots$. In order to estimate what set of errors possibly occurred, we need to determine the eigenvalue of each generator of the stabilizer. The eigenvalue of a generator M_i will be $(-1)^{f_{M_i}(E)}$. This process gives us the error syndrome. Using this error syndrome we can infer what errors are most likely.

We use one ancilla bit each to measure the eigenvalues of each of the generators. The

ancilla bits are initialized to the $|0\rangle$ state and then the Hadamard transform is applied. Each ancilla bit controls the application of one of the generators M_i. The ancilla bit undergoes a Hadamard transform again and it is measured in the $\{|0\rangle, |1\rangle\}$ basis.

1. Quantum Viterbi Algorithm

Unlike as in the block code case, we do not use all the error syndromes at the same time. Instead we borrow the idea behind classical Viterbi algorithm and build the error vectors incrementally. At each step, we collect a fixed number of error vectors of certain length that are the most likely.

Before we proceed to the quantum Viterbi algorithm (QVA), we introduce the term: *memory state*. A *memory state* is given by the error operators acting on the last m qubits being considered in step i, where m is the block overlap. Two error vectors which contain the same error operators acting in last m positions are said to end in the same memory state. Hence, there exist 4^m possible states. The *partial path metric* is the weight of the error operators up to step i.

Following are the steps involved in the QVA:

1. *Initializing step:*

   (a) Compute the first l error syndromes, corresponding to the l initializing generators.

   (b) Determine the list of $4^m/2^l$ error vectors, $\mathcal{E}_0$, of length m that are compatible with the l syndromes, where m is the block overlap.

2. *For i=1 to t, where t is the number of the input blocks:*

   (a) Compute the error syndromes corresponding to the generators $M_{d(i-1)+1}$ through $M_{(di)}$, where $d = n - k$.

(b) List all possible extensions of errors in $\mathcal{E}_{i-1}$ that are compatible with the error syndromes computed in the previous step such that the errors now act only on the qubits from positions 1 to $(m + ni)$. This new list is denoted by $\mathcal{E}_i$.

(c) Compute the weights of all errors ending in each of the 4^m states.

(d) For each state, pick the error vector with the least weight.

(e) This gives us the final list of most likely errors for step i, $\mathcal{E}_i$, that contains 4^m error vectors.

3. *Terminating step:*

(a) Compute the t error syndromes, corresponding to the t terminating generators

(b) Determine the list of errors from the previous step that are compatible with the t syndromes. This list is denoted by $\mathcal{E}_\infty$

(c) Pick the error with the least weight among the ones listed in $\mathcal{E}_\infty$

Let us take the standard form of the generators obtained in the previous section to understand the QVA. The code has rate 1/5. d is 4. It has $l = 1$ initializing generators and $q = 4$ terminating generators.

During the initialization step, we measure the error syndromes corresponding to the initializing generators. In our case, $l = 1$ and we measure the syndrome of M_0 given as

$$M_0 = Z\ X\ I\ I\ I\ \ldots$$

M_0 acts on $m = 2$ qubits. We have 4^2 possible error vectors acting on 2 qubits. But only $4^2/2 = 8$ such vectors are compatible with the error syndromes. We list these 8 error vectors as $\mathcal{E}_0$.

In step 2, we start with $i = 1$. We compute the error syndromes of generators M_1 through M_4. We use these syndromes to generate a list of compatible error vectors that act

on qubits 1 to 7. This list represented using $\mathcal{E}_1$ contains error vectors that are extensions of the error vectors in $\mathcal{E}_0$. These vectors are compatible with syndromes M_0 through M_4.

The weight of each of the error vectors in $\mathcal{E}_0$ is computed. As in the classical Viterbi algorithm, we need to keep the number of likely error vectors under control to avoid an exponential blowup. In the classical Viterbi algorithm, at each state, which contains a particular memory configuration, a local decision is made. Among all the sequences ending up in that state, only the one with the minimum weight is picked and the others are rejected. If there is a tie between two or more sequences, then one is picked at random. So after each step we have a list of as many most likely codewords as there are states. We use the same idea to maintain a fixed size list in the quantum case.

For our example, we consider all the error vectors ending in each of the $16 = (4^2)$ states and for each state, we pick the error vector with the minimum weight. If there is a tie between two or more vectors, then we pick an error vector at random. We then compile the final list $\mathcal{E}_1$ which contains 16 most likely error vectors, each of which ends with a different state. We follow this procedure until we reach the terminating generators.

In the terminating step, we compute the syndromes for the generators M_∞^j,i.e., M_∞^1 through M_∞^4. Using these syndromes we again select compatible errors, $\mathcal{E}_\infty$, from the list of errors obtained in the previous step. We pick the error vector from $\mathcal{E}_\infty$ with the minimum weight, which gives us the most likely error given the syndromes for the received sequence.

Once we obtain the error that has most likely occurred, we apply the inverse of this error on the received codeword to fix it. To decode the codewords we run the encoding circuit backwards.

2. Windowed Quantum Viterbi Algorithm

Following the above algorithm, we obtain the most likely error only after we take the last syndrome into consideration. Hence, we have to, in principle, wait until the entire information sequence is received to estimate the error that has occurred. There exists a variation of the classical Viterbi algorithm, called the *windowed Viterbi algorithm*, which is based on the fact that the likely codeword sequences selected at a given step coincide with the most likely sequence except on the last few positions. In windowed Viterbi algorithm, we need not wait until the end of the transmission, but only up to a certain latency L.

In our case, after we follow the quantum Viterbi algorithm for more than L steps, we can make a decision as to what error has most likely occurred in step $(L - i)$ when i is the index of the current step. Following the same procedure in our case, we can now estimate the most likely error online, without having to wait for the end of transmission.

CHAPTER V

CONCLUSIONS AND FUTURE WORK

In this thesis, we developed the general theory for quantum convolutional stabilizer codes. The stabilizer formalism of the quantum convolutional codes was discussed along with the properties and the structure of the stabilizer. Various distance properties of the QSC codes were also defined.

The special structure of the stabilizer of the QSC codes allows for online encoding and decoding. We showed how the generators of the code can be used to construct the quantum circuit to compute the codewords. This quantum circuit can be used to perform online encoding. The original generator block is used to construct a gate array and this gate array is shifted to encode the succeeding blocks of qubits. One of the main challenges we faced while developing the theory for constructing the encoding circuits was that, the generators after being converted to standard form had to still commute with the generators of the preceding and succeeding blocks. We overcame this problem by introducing the pseudo-standard form. To obtain the pseudo-standard form, we need not transform the generators by reordering the qubit positions. Hence, we always maintain the commutative nature of all the generators.

We also described the quantum analogue of Viterbi algorithm to estimate the most likely error for a given received sequence. While the idea behind the quantum Viterbi algorithm (QVA) was similar to its classical version, it required a somewhat new approach. Here we were dealing with error vectors instead of sequences of bits. Also, the memory states are defined differently. Using QVA we can begin decoding without having to wait for the entire received sequence. At each state, the number of error vectors is kept fixed by considering only the ones with the least weight, thereby, avoiding an exponential blowup of the list of error vectors. In the final step, QVA outputs the error vector that has most

likely occurred, given the error syndromes of the received sequence. The complexity of QVA grows linearly with the number of encoded qubits. We also discussed another version of QVA called the Windowed QVA. Windowed QVA enables us to fix the error after a certain delay L without having to wait for the entire received sequence to be decoded.

We took the help of an example rate 1/5 QSC code to explain the encoding and decoding processes. In general, any set of generators that satisfy the properties required for QSC codes, can be used to define a quantum convolutional code. It would be helpful if the necessary theory to compute the weight enumerators for the QSC codes was developed. It would help in understanding the distance properties and performance of practical codes better.

REFERENCES

[1] P. W. Shor, "Scheme for reducing decoherence in quantum computer memory," *Phys. Rev. A*, vol 52, pp. R2493-R2496, October 1995.

[2] A. M. Steane, "Error correcting codes in quantum theory," *Phys. Rev. Lett.*, vol. 77, pp. 793-797, July 1996.

[3] A. R. Calderbank and P. W. Shor, "Good quantum error-correcting codes exist," *Phys. Rev. A*, vol. 54, pp. 1098-1105, August 1996.

[4] A. Steane, "Multiple particle interference and quantum error correction," *Proc. Roy. Soc. Lond. A*, vol. 452, pp. 2551-2577, November 1996.

[5] E. Knill and R. Laflamme, "A theory of quantum error-correcting codes," *Phys. Rev. A*, vol. 55, pp. 900-911, February 1997.

[6] C. H. Bennett, D. P. DiVincenzo, J. A. Smolin and W. K. Wooters, "Mixed state entanglement and quantum error correcting codes," *Phys. Rev. A*, vol. 54, pp. 3824-3851, November 1996.

[7] D. Gottesman, "Class of quantum error-correcting codes saturating the quantum hamming bound," *Phys. Rev. A*, vol. 54, pp. 1862-1868, September 1996.

[8] A. R. Calderbank, E. M. Rains, P. W. Shor, and N. J. Sloane, "Quantum error correction and orthogonal geometry," *Phys. Rev. Lett.*, vol. 78, pp. 405-408, January 1997.

[9] A. M. Steane, "Enlargement of Calderbank-Shor-Steane quantum codes," *IEEE Trans. Inf. Theory*, vol. 45, pp. 2492-2495, November 1999.

[10] A. Y. Kitaev, "Quantum error correction with imperfect gates," in *Proc. 3rd Int. Conf. of Quantum Communication and Measurement*, New York, May 1997, pp. 181-188.

[11] D. Aharonov and M. Ben-Or, "Fault-tolerant quantum computation with constant error rate," in *Proc. 29th Ann. ACM Symp. on Theory of Computing*, New York, May 1997, pp. 176-188.

[12] E. Knill and R. Laflamme, "Concatenated quantum codes," quant-ph/9608012, August 1996. Available at http://xxx.lanl.gov/PS_cache/quant-ph/pdf/9608/9608012.pdf.

[13] P. W. Shor, "Fault-tolerant quantum computation," in *Proc. 37th FOCS*, Los Alamitos, CA, March 1996, pp. 56-65.

[14] J. Preskill, "Reliable quantum computers," *Proc. R. Soc. Lond. A*, pp. 454-385, August 1997.

[15] D. Gottesman,"A theory of fault-tolerant quantum computation," *Phys. Rev. A*, vol. 57, pp. 127-137, January 1998.

[16] A. M. Steane, " Efficient fault-tolerant quantum computing," *Nature*, vol. 399, pp. 124-126, May 1999.

[17] D. Gottesman, "Fault-tolerant quantum computation with local gates," *J. Modern Optics*, vol. 47, pp. 333-345, February 2000.

[18] H. F. Chau, "Quantum convolutional error correcting codes," *Phys. Rev. A*, vol. 58, pp. 905-909, August 1998.

[19] H. F. Chau, "Good quantum convolutional error correction codes and their decoding algorithm exist," *Phys. Rev. A*, vol. 60, pp. 1966-1974, September 1999.

[20] H. Ollivier and J. -P. Tillich, "Description of a quantum convolutional code," quant-ph/0304189, April 2003. Available at http://xxx.lanl.gov/PS_cache/quant-ph/pdf/0304/0304189.pdf.

[21] D. Gottesman, "Stabilizer codes and quantum error correction," California Institute of Technology Ph.D. Dissertation, Pasadena, CA, May 1997.

VITA

Neelima Chinthamani was born on June 2, 1980 in Dornakal, India. She received a B.S. degree in computer science and information technology from the Jawaharlal Nehru Technological University, India, in July 2001. She entered Texas A&M's graduate program in the Fall of 2001. She interned in Dell Inc., Austin, Texas, during the summer of 2003. She was funded by Dr. Andreas Klappenecker for the entire duration of her graduate studies. Her research interests include quantum error correcting codes and fault tolerant quantum computing. This thesis marks the end of her long trek through graduate school. Her permanent address is Plot No. 58, Srinivasa Colony, Swaroop Nagar, Uppal, Hyderabad, Andhra Pradesh, India 500039.

The typist for this thesis was Neelima Chinthamani.